Probabilistic Complex Event Triggering

Daisy Zhe Wang
Eirinaios Chrysovalantis Michelakis
Liviu Tancau

Probabilistic Complex Event Triggering

Daisy Zhe Wang, Eirinaios Michelakis, and Liviu Tancau

Computer Science Division
University of California at Berkeley, Berkeley CA 94720, USA
{daisyw,ireneos,tancau}@cs.berkeley.edu

Abstract. Recently, wireless sensor devices have been widely deployed in various application settings (including environmental research, control systems, etc.). Because of the inherent unreliability of sensor readings, any kind of reasoning in sensor environments needs to carefully account for noise. The key goal of PCET is to build an infrastructure that can automatically infer and reason about the probabilities of triggered events, using a principled probabilistic model for the underlying sensor data. Through such probabilistic reasoning, PCET can incorporate uncertainly factors and make finer – grain decisions on event occurrences. This is achieved through the use of a Bayesian Network to directly model and exploit correlations across different sensors and the definition of a complex – event language, which allows users / applications to create hierarchies of higher-level events. As experimental results verify, PCET simplifies the development process and boosts the efficiency of any system dealing with inherently uncertain data streams.

1 Introduction

Recently, the research community has become a witness of the rising interest for the efficient monitoring and analysis of data that arrive continuously in high speed streams. This tension gave birth to the field of Data Stream Management, which incorporates techniques from traditional Database Management Systems and Data Mining, in order to efficiently process streaming data. Event monitoring and triggering systems were quite thoroughly studied for over a decade now, mostly in the field of Active Databases [32], aiming in providing applications the necessary means for automatic reaction in presence of certain events. Transferring the notion of events to the data streaming paradigm, it is not surprising that recent efforts tried to apply the ideas of complex event triggering to data streaming applications, while at the same time recognizing the peculiarities of the latter systems, such as scalability issues and the expressiveness of the event algebras to be used in order to provide the semantics that a streaming application would require from the event triggering system [6].

A rather important issue that so far has escaped the immediate attention of the event monitoring systems community is that of the inherent uncertainty that data (and in particular events) are characterized from. Especially for data streaming systems, for which a large class of applications is fed by data coming from unreliable sensor readings, the need to incorporate this notion of uncertainty into the event processing engine seems quite important, as it would allow for more realistic inference mechanisms that could identify the subset of events that seem *more likely* to have occurred, given a set of noisy readings. Obviously, noisy or missed sensor readings may mistakenly trigger events, and the impact of such false positives / negatives could be high, depending on the application.

This probabilistic interpretation of data has attracted recently the interest of the Database community [8, 14, 25]. The real problem to face is how can we, as realistically as possible, model the correlation of the data objects (or in the case of a complex event monitoring system - events) of our world. One of the most common approaches is to ignore the problem in its entirety and assume complete independence between the data. Other approaches model the data objects as random variables obeying a probability distribution (with the most prominent example of BBQ [8], where the readings coming from the sensornet are assumed to follow a multivariate Gaussian distribution, the parameters of which are subject to

This paper was produced as a partial fulfillment of the project requirements for the courses: CS262A, "Advanced Topics in Computer Systems" and CS281A, "Statistical Learning Theory - Graphical Models". Daisy Zhe Wang and Eirinaios Michelakis attended both CS262A and CS281A, while Liviu Tancau attended CS262A only. Fall Semester, 2005.

be learned from the data). In a more realistic setting, we would expect to be able to identify the correlations between the objects of our data model, by making as few independence assumptions as possible, in order to increase our confidence on the probabilistic inference we would like to perform on them.

Another very important requirement that event monitoring systems have to satisfy, in an effort to ease the development of applications that are the final recipients of the triggered events, is the expressiveness of the language constructs with which events can be specified. A common approach that is followed to increase the expressiveness of the language semantics is the definition of events in a hierarchical manner. The way that a so called *complex* event is comprised of its constituent, or *base–level* events, is formalized by the event algebra used. For an event triggering system based on a streaming data model, the base events coincide with tuples from the streams that feed the system (in a sensornet environment these correspond to the actual sensor readings). For an illustration of a complex event architecture see [27].

This paper describes an approach that attempts to resolve the problem of robust event detection and triggering in the presence of noisy inputs, while at the same time offering an expressive language through which probabilistic event hierarchies can be easily specified. The end result of this effort is "PCET", a prototype of an event processing system, able of performing probabilistic inference on complex events defined in its context. To the best of our knowledge, exploitation of probabilistic models for use in event processing systems has not been sofar explored in the relative literature.

To support probabilistic inference, PCET maintains a graphical model, namely a Bayesian Network that is derived from noisy sensor outputs, collected over a period of time, and from information about the sensitivity of the sensors, provided by the manufacturer. Thus, guided by the sensing quality of the nodes in a sensornet deployment, the system can infer correlations between them by directly observing their output. By using this model, our confidence on the occurrence of an event in a given point in time can be increased or decreased, based on the observation of the events that are correlated to it, thus enhancing the robustness of the monitoring process with respect to the noise induced in the raw data readings.

To address the expressiveness issue, PCET provides a language that enables the end user to formulate hierarchies of event definitions, augmented with probability predicates that control the triggering of those events to the application layer. Therefore, for a higher level event to be triggered it is not enough that the streamed values from the sensors simply verify equality or membership predicates on their constituent lower level events; it is also required that the probabilities of those events occurring accord with the probability thresholds associated with them. This approach enables the specification of events at different abstraction layers, thus providing the attractive *modularity* property of encapsulating the details of the "probabilistic event streams" flowing through the system into lower building blocks, which drastically simplifies application development.

We evaluate our system in the context of the DIGITAL HOME (DH) effort at Intel Research Berkeley. We define the event hierarchy of an example application that keeps track the whereabouts of the tenants of a Digital Home, on a small scale sensor network deployment. The experimental results indicate that the task of complex event processing can be significantly improved by the modeling of correlations between the participating sensors, as opposed to the naive approaches of regarding each sensor independent of its peers, especially in an unreliable or noisy environment.

The paper is organized as follows. Section 2 gives a brief overview of the related work on the areas of probabilistic data management and event processing. Section 3 introduces the main architecture of PCET, along with the graphical modeling that enables probability inferencing and the complex event definition language. A brief introduction to Bayesian Network learning and inferencing methods is provided at section 4. Section 5 assesses the implementation issues regarding the prototype of the system. Section 6 describes the use case scenario in which PCET was evaluated and reports the experimental results which we collected, while section 7 concludes, with references to future directions.

2 Related Work

In this section we provide a brief overview of the published work, closely related to our approach. We will discuss the course that event processing followed during the last fifteen years and its emergence in modern application scenarios. Then, we will go through the most important advances in Probabilistic Data Management, a field that made its first appearance during the early nineties, but recently has attracted a lot of attention by the community. Finally we will mention a couple of approaches that lie in the realm of probabilistic data management, which use probabilistic models to infer activities in the context of the "Digital Home" vision.

2.1 Complex Event Processing

Many research initiatives have looked in the past at the development of systems able to respond on the occurrence of certain events in a timely manner. In the context of Data Management systems, this idea was explored particularly by the Active Database community [31]. Unlike traditional databases, that are used only for storing and processing of historical data, thus displaying a *passive* behaviour, Active Database Systems support triggering to the application level of certain events, that can be either simple data insertion, modification, transaction commit and failure operations, or more complex events based on a set of more simple event definitions as the ones described above, using *event algebras*. The most commonly used form of triggers (or rules) are the so called ECA (Event-Condition-Action) Rules. For a comprehensive analysis of the semantics of complex event algebras in the context of the Active Database literature, the reader is referred to [32].

The recent interest of the research community in efficient data streams management and the broadening of our understanding in distributed systems has very naturally led to the emersion of applications that monitor high speed streams, in order to be able to react in *real-time*, whenever certain *patterns* in data emerge. The need of event triggering support in these systems was evident from the very early stages of their development, and the maturity of the complex event algebras at the time assisted in combining those two worlds. Of the most recent contributions in this field, Demers et al. [6] propose CEASAR, an event algebra for the data streaming paradigm, which has well-defined semantics and provides support for parameterization and aggregation. Their approach is tested on CAYUGA, a prototype event stream processing system, which enables high-speed processing of large sets of queries expressed in the CEASAR algebra. In the context of sensor networks and the HiFi project in particular [11], Rizvi et al. [27] propose a novel architecture for processing complex high level events, defined on the basis of data outputted by receptor devices at the edges of High Fan-In systems.

2.2 Probabilistic Data Management

Probabilistic database systems, along with proposals for new data models - extensions to the dominant Relational Model, able to capture incomplete or imprecise data, came into the foreground since the early nineties. [19] is a rather simplistic example of such a system. As the work in this field draw attention, more sophisticated models were appearing, that associated probabilities with values of attributes and proposed ways to perform inference on them [1]. More recently, Siciu in [4] and in a series of papers that followed, studies the problem of efficiently answering queries on probabilistic databases, returning a list of ranked tuples by their probabilities. A systematic effort is made to revisit the semantics of traditional relational operators, in order to enable them to handle values with probabilities. A more theoretical approach is followed by the TRIO project [29], where two layers for managing uncertain data are defined: a complete logical model, and one or more incomplete views of it, the*working* models, that may lose some information, but are easier to understand and query.

Our effort has been mainly inspired by the line of work done in [10, 8]. Faradjian et al. ([10]) introduce GADT (the Gaussian Abstract Data Type), a new object-relational data type that models physical (continuous) data coming from unreliable sensors as gaussian probability distributions. They propose a theoretical framework for studying the probability space of abstract data types and they describe a prototypical implementation of GADT as an extension to the Cornel PREDATOR object relational DBMS.

On a more applied ground, Deshpande et al. propose in [8] a robust sensornet query processing architecture (BBQ) that uses statistical models (specifically a multivariate gaussian distribution) to handle imprecisions in sensor readings and approximate query answering within certain probabilistic confidences. The main difference with our approach is that we try to capture the correlation between different sensors using a more robust and intuitively reasonable model (a Bayesian Network). We expect that modeling the correlation of a large number of random variables through a single multivariate distribution imposes a lot of restrictions on the relations of those variables, both semantic (the physical qualities to which they correspond) and representational (discrete and continuous-valued sensors could coexist in the same sensornet deployment), and thus it loses somehow in generality. Furthermore, BBQ is oriented in answering one-shot queries over the network, while the event monitoring framework we are examining corresponds to continuous query processing.

PCET does not represent the first effort of employing graphical models in data management research.

Literature that lies in the intersection of AI and theoretical data management is rich in publications that borrow such notions from the Statistical Learning theory (e.g. [14, 28]). The most notable example is perhaps the work on Probabilistic Relational Models (PRM's) [14]. These models capture the uncertainty over attributes of objects in a domain and the [in]dependence of different level of classes of objects. The means to achieve that is by performing Bayesian Network inference. The above paper describes how can a PRM be learned from an existing relational database, using structure and parameter learning techniques in a similar fashion as the one introduced in this paper. Our system extends this approach and specializes it for the complex event processing paradigm. For a comprehensive overview of the merging of AI modeling techniques, learning and probabilistic inference, the reader is referred to [25].

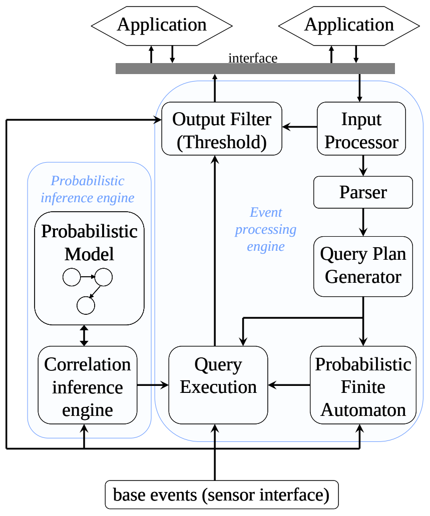
Application
Application
interface
Output Filter (Threshold)
Input Processor
Probabilistic inference engine
Parser
Event processing engine
Probabilistic Model
Query Plan Generator
Correlation inference engine
Query Execution
Probabilistic Finite Automaton
base events (sensor interface)

Fig. 1. The general architecture of PCET.

2.3 Applications

The "Digital Home" vision that powered our efforts in the context of this project is becoming recently an increasingly active research topic, as it merges interesting material from various disciplines (Human Computer Interaction, Pervasive Computing, Machine Learning, Statistical Modeling, Sensor Networks, Data Management to name a few). Below we present a small number of projects that explore this agenda. The University of Texas at Arlington, through the MavHome project, attempts to predict the actions of the home's inhabitants by using a variation of Hidden Markov Models which is based on unsupervised learning techniques [26]. Intel Research Seattle [24] runs a similar project that attempts to infer human activities from the home occupants interactions with objects. With the help of NLP techniques and the web as a secondary source of information, they create activity models which are associated with the probabilities of certain objects being involved in those activities. At Georgia Institute of Technology, the Aware Home project [18] proposes various tracks of ubiquitous sensing and inferencing applications in the home environment; as an example, the "Smart Floor" initiative tries to recognize the individuals in the house by the pattern of their footsteps, using Hidden Markov Models to create ground reaction force profiles for each occupant.

3 Architecture

In this section we will describe the general architectural components of PCET, emphasizing on the functionality that a probabilistic complex event processor should export to the application layer. We will address the expressiveness characteristic that such a system should support, by elaborating on the specifics of an event language designed for our system, to enable the efficient definition of probabilistic events hierarchies. Finally we will present the probabilistic framework of our approach, namely a Bayesian Network topology, suitable for the context of event monitoring for sensor network deployments.

3.1 General Architecture

The general architecture of PCET is illustrated in figure 1. As indicated, it is composed of four basic elements: an Application Layer Interface, a Sensor Layer Interface, an Event Processing Engine, and a Probabilistic Inference Engine. In the following paragraphs we provide a detailed description of these components and the interactions that occur between them.

Application Layer Interface The Application Layer Interface allows PCET to communicate with the user applications written "on top" of it. Through this interface, applications can define new high-level events based on sensor readings and / or simpler events that are already in the system (the definition language is fully described in section 3.2). Applications can also subscribe to events of interest, providing PCET with a notification threshold. PCET continuously evaluates the probabilities for all events of interest and whenever the probability of such an event exceeds the threshold provided by an application, that application is notified of the (probable) occurrence of the event through an asynchronous message containing the event name and the computed probability. Naturally, applications can also unsubscribe from events that are no longer interesting, or even delete them.

Sensor Layer Interface The input of PCET is comprised of data streams generated by sensors. However, due to the heterogeneity of sensor interfaces, we cannot connect PCET directly to the raw data streams. Instead, we require the addition of an external data processing layer that gathers data from the hardware devices and converts it into the format that PCET expects (probability distributions). This sensor layer must be able to service data requests (pulls) made by our system at any time, possibly having to estimate readings for sensors that cannot provide data at that moment. It could also perform a number of other useful operations such as data cleaning and data aggregation.

Event Processing Engine The Event Processing Engine is responsible for compiling the definitions of events of interest into a set of machine level constructs that are continuously executed to evaluate the probabilities of the events (and notify applications if necessary). In a sense, its operation resembles that of a DBMS query execution engine, but in a different context.

The Event Processing Engine handles application requests using an Input Processor, which sends new event definitions to the parser and subscribe / unsubscribe requests to the event output (thresholding) unit. The parser validates event definitions and converts them into some internal representation. The Query Plan Generator takes the parsed event definitions and breaks them down into base events (i.e. expressions on the incoming sensor data), assembling a plan for evaluating the high level event probabilities. The Query Execution unit then follows the plan, computing joint probabilities for the sensor readings involved in the query (with assistance from the Probabilistic Inference Engine) and marginalizing them according to the logical predicates in the event definition. The result of execution is a set of (event name, probability) tuples, which are sent to the thresholding output filter (which may in turn forward them to applications).

In order to support the time sequencing predicates of the SIMPLE EVENT LANGUAGE (see section 3.2), we also capture the history of certain queries through a Probabilistic Finite Automaton that cooperates with the Query Execution component.

Probabilistic Inference Engine The main goal of the Probabilistic Inference Engine is to accurately compute joint probability distributions of several sensor readings using the correlation information stored in a probabilistic model of the sensors. It is also responsible for transforming observed values (sensor readings) into estimates of the real values measured. Sofar, the model that enables both of these tasks is induced by the data in an off-line manner. A more detailed description of the graphical model used is provided in section 3.3.

3.2 Event Definition Language

Applications can define complex events for PCET to monitor using the SIMPLE EVENT LANGUAGE (SEL). SEL has very few constructs (see figure 2 for the complete grammar), which makes it easy to learn, but is expressive enough to allow the specification of many interesting events. The features of the language are discussed in the following subsections.

Sensor readings The simplest event that one can write involves comparing the output of a sensor with some value, using standard arithmetic comparators (e.g. `Sensor1>0`). It is also possible to compare two sensor readings (e.g. `Sensor1=Sensor2`) or two constant values, though the latter has little usefulness. Certain sensors have labels (value aliases) associated with them. These labels can be used instead of constants in a comparison expressions and their meaning depends on the sensor involved in the comparison (e.g. `IDSensor=Alice` is equivalent to `IDSensor=1` if `IDSensor` exports a label 'Alice' with value 1).

The semantics of "`sensor comp value`"[1] is simply an event with probability $P($`sensor comp value`$)$, where `sensor` is the actual value of the `Sensor` RV.

Boolean Expressions As is common in most languages, SEL allows basic expressions to be combined logically using Boolean operators (conjunction and disjunction). However, unlike most languages, the value of a Boolean expression is not computed by evaluating the parts first. The semantics of "`(S1 comp v1)`

[1] `comp` corresponds to a comparison operator (`<, >, =, !=`)

```
           (Event) E ::= event_id = e
      (Expression) e ::= E | e '&' e | e '|' e | '!' e | e ';' e | e ';;' e  |
                         '{' e '}' time | v ('<' |  '>' |  '=' | '!=') v
(Random Variable) v ::= const | label | sensor_id | v ('+' | '-') v
```

Fig. 2. The grammar of SEL in EBNF format.

`op (S2 comp v2)`" is an event with probability $P($`s1 comp v1` $\wedge$ `s2 comp v2`$)$. Generally speaking, this requires the evaluation of the joint probability distribution of `S1` and `S2`, and the summation of probabilities corresponding to all tuples (s1,s2) that satisfy the boolean expression. This argument extends to an arbitrarily long concatenation of logical expressions.

Hierarchical Events An event definition can refer to previously defined events, treating them as expressions (e.g. `Intruder = Entry & !ConfirmedID`). The semantics of higher level events are defined in terms of substitution, i.e. recursively inlining the definitions of lower level events wherever their name appears until the only identifiers that appear in the definition are sensor names.

Time bracket The definitions described up to now deal only with current events, i.e. events that occur at the time when the query is evaluated. The time bracket construct allows us to specify events that may have occurred at some time in the past, within a given time frame that ends in the present. For example, "`{DoorBell=1}5min`" is an event that indicates whether the doorbell rang during the last 5 minutes (from the time of execution). When a time bracket surrounds a logical expression such as "`{A & B}1min`", the meaning is "A and B occurred simultaneously at some point in the last minute". To specify that A and B both happened in the last minute but not necessarily at the same time, one can write "`{A}1min & {B}1min`".

Sequencing It is often necessary to express that certain events occurred in a certain order, which the time bracket does not handle. This is accomplished in SEL using the sequence operator (`;`), as in "`A ; B`". The sequence operator uses an implicit time window, which could be a system-wide or application-specific parameter. To override the default time window, one can use the time bracket, as in "`{A;B}30sec`". The semantics of a "`A;B`" is an event with the probability that A happened at some time and B happened some time later. If temporal independence is assumed, this can be computed by summing the probabilities $P(a@t_1 \wedge b@t_2)$ for all $t_1 < t_2$ in the time frame.

Stride While the sequence operator does convey the meaning of one event happening after another, it does not allow us to specify how far apart in time they were. "`A;B`" implies that the timing difference of the to event could be as small as the sampling period or as large as the time frame. In contrast, by using the stride operator we can define a sequence "`A;;B`" in which the events happen precisely one time frame apart. As before, the time frame can be made explicit or it will default to some value. This allows us to define an event such as "`{BobEnters ;; TV=off}2min`" which will trigger if the TV is (still) off 2 minutes after Bob entered the house. Note that the sequence and stride operators can be combined in order to specify more general timing constraints, such as "Alice brushed her teeth 3-5 minutes after waking up": "`{{AliceWake;true}2min;;ToothbrushNoise}3min`". Here we had to introduce the dummy event "`true`", which has probability 1 at all times; it could also be encoded as "`1=1`".

By combining the above constructs, we believe that one can encode a great variety of events that refer to both space (inherent in the sensor locations) and time (with variable time frames and sequencing operators). The main challenge is in evaluating them in a realistic way, taking into account correlations. The rest of the paper provides the theoretical background and our design decisions in terms of implementing the framework we have thus far described.

3.3 Probabilistic Models

In this section we present the graphical model on which our approach is based. Our effort has been to capture in a unified way both the correlations among the sensor data and the sensors' noise patterns, which we regard as an external parameter of our system. Following, we examine these two modeling decisions separately.

Modeling Sensor Noise As pointed out in section 1, sensors, especially wireless ones, are subject to high levels of noise, which could be attributed to various reasons (e.g. device's sensitivity, electromagnetic interference, fluctuations in the power levels, environmental conditions in general, etc.). In reality, noise patterns on different sensors might be correlated; however, for the sake of simplicity, we assume that noise patterns on different sensors are independent with each other. Thus, each sensor's output can be modeled as a random variable, which is been diluted by the presence of noise.

Let X be a random variable, corresponding to the real value of the measured physical quantity by a specific sensor, taking values by some distribution that governs that physical quantity, $N(X)$ the induced noise and X' a random variable corresponding to the sensed quantity. Then, the following relationships hold:

$$X' = X + N(X) \tag{1}$$

$$E(X') = X + E(N(X)) \tag{2}$$

$$Var(X') = Var(N(X)) \tag{3}$$

Thus, given the real value, the distribution of the sensor reading is shifted by the amount of noise for that particular value of X, $N(X)$. We model noise as a the conditional probability $p(X'|X)$, where X is the real value and X' is the sensed value. We expect that this conditional distribution will be provided by the sensor manufacturer as part of device's specifications. For example, a motion sensor manufacturer can experiment on the number of times the sensor reports correctly or incorrectly movement in its range, being therefore able to synthesize, after a statistically sufficient number of the experiments, the conditional probability table discussed above.

For the current implementation of PCET, we assume this conditional distribution to be fixed. However, we are aware that in reality, the reliability of the readings varies with respect to the environmental conditions under which the sensor operates. We expect that in the future, as more sophisticated applications come into the foreground that require reliability of the sensor readings, this issue of reliability guarantees will have to be addressed. The Database group of the University of California at Berkeley has already started the development of an abstraction layer, that is tuned to a specific sensor like a virtual device driver, and filters its output, in order to provide confidence levels of the sensed values to the end user. Preliminary results of this effort seem quite encouraging ([16]).

Modeling Correlations In practical situations, one would expect that the data collected from a sensor network deployment to be often correlated. Many reasons can contribute to this phenomenon, among which the locality of the sensors – sensors in close proximity to each other could capture the same event at a specific point in time.

Previous work [8] has exploited the spatial correlations between the sensors in a sensornet deployment, to answer approximate queries without having to consult all the sensors. However, noisy or missed sensor readings may mistakenly trigger events, and the impact of such false positives / negatives could be high, depending on the application. Therefore, an application could react differently if it knew the confidence of the occurrence of an event being reported by a complex event triggering engine. In addition, if the existing correlations between the sensors were to be exploited, the confidence of an event could be enhanced or lowered, based on the confidence values of the other sensors in the deployment. In order to achieve the these goals, we need a model to capture the correlations between the sensors and output the probability of certain events happening (probabilistic queries or high level event definitions), while at the same time interpreting the sensed values (X') according to the noise model the sensors obey.

Bayesian (or Belief) Networks can be regarded as a natural choice for this task, as they have been widely used for modeling uncertainty in knowledge representation. This approach uses probability theory to reason with uncertainty, by explicitly representing the conditional dependencies (correlations) between the different entities, while providing an intuitive graphical representation of the mathematical model.

By using the assumptions regarding the noise model that we described in the previous paragraph, we propose a BN model that combines the conditional dependencies of the real data values, along with the noise patterns that are inflicted on the sensors' outputs. The product of this combination is illustrated in figure 3. This example represents a model of four sensors in a particular deployment scenario. Nodes X_1 through X_4 correspond to the real values of the measured physical quantities and nodes X'_1 through X'_4 to the sensed (noisy) readings. As we stated in the previous paragraph, we assume that X'_i are in-

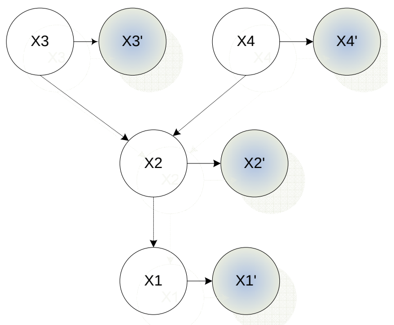

Fig. 3. An example BN for a particular deployment that involves four sensors. Nodes X_1 through X_4 correspond to the real values of the measured physical quantities and nodes X'_1 through X'_4 to the sensed (noisy) readings.

dependent of all other nodes in the network, given X_i.

The correlations between the variables (X_i) need to be learned from observations of the real values of the sensed quantities. Since the set of observations usually available contains only the sensors readings and not the actual values, we have to use learning techniques, able to infer both the structure and the parameters of the BN from those *incomplete* observations. Thus, nodes X_i are in essence *hidden*, while X'_i are *observed*. To address this problem, we experimented with two approaches.

The first makes use of structure learning algorithms for complete data, to learn the correlations between the random variables that correspond to the actual values (X_i's), by using as input the noisy sensor readings. Then, the structure of this partial inferred network (dotted edges in figure 3) is enriched by adding the nodes corresponding to the sensed readings (X'_i's), with the edges that correlate each X_i with its noisy counterpart (X'_i). After that, the parameters of the network are learned using the EM algorithm. The second approach uses SEM, an algorithm capable of inferring both the structure and the parameters of a BN, using partial observations. For a brief overview of the learning algorithms used, see section 4.

4 Bayesian Network Learning and Inferencing

In this section we will attempt a brief overview of Bayesian Network (BN) learning and inferencing algorithms, with more emphasis to those that were used in this project. Specifically, we will go through the learning tasks that comprise the inference of a BN from a training data set, namely *structure* and *parameter* learning (sections 4.1 and 4.2) and we will describe the inference algorithm we chose to use (section 4.3), explaining the reasons for this particular choice.

4.1 Parameter Learning

The task of parameter learning involves the estimation of the parameters of the graphical model (i.e. the conditional distribution of a node with respect to its parents), from a number of observations that correspond to a subset of the model's nodes. For the purposes of this project we are experimenting with models with discrete random variables.

Let $G = (V, E)$ be a directed graph. We associate a random vector of discrete valued variables X, where its components are indexed by the nodes in the graph. Thus, X_u denotes the random variable associated with node $u \in V$, and x_u denotes a realization of X_u. By taking advantage of the conditional independence assumptions encoded in the graph's structure, we can factorize the overall probability associated with G, as a product of local conditional probabilities of the nodes of the G given their parents. Thus, we can write:

$$p(x_V|\theta) = \prod_{u \in V} p(x_u|x_{\pi_u}, \theta_u), \tag{4}$$

where θ_u is the parameter vector associated with variable X_u and π_u is the set of parental nodes of X_u. By using a tabular joint probability distribution representation, we associate a separate parameter for each possible joint configuration of a node and its parents. Thus, we define $\theta_u(x_{\phi_u})$ to be the local conditional probability table of node u, where ϕ_u denotes the family of nodes associated with u. Therefore, equation (4) can be written as a product of potentials:

$$p(x_V|\theta) = \prod_{u \in V} \theta_u(x_{\phi_u}), \tag{5}$$

In the case that learning is performed on a *complete* set of independent and identically distributed (*iid*) observations, that is sample values for all $X_u, u \in V$ are present, the set of parameters $\hat{\theta}$ can be learned by computing maximum likelihood estimates from the training data. We can calculate then the complete log likelihood of the data as:

$$\begin{aligned}\log p(D|\theta) &= \log\left(\prod_n p(x_V, n|\theta)\right)\\ &= \sum_n \log\left(\prod_{x_V} p(x_V|\theta)^{\delta(x_V,x_{V_n})}\right)\\ &= \sum_{x_V} m(x_V)\log p(x_V|\theta)\\ &= \ldots =\\ &= \sum_{u\in V}\sum_{x_{\phi_u}} m(x_{\phi_u})\log\theta_u(x_{\phi_u}),\end{aligned} \tag{6}$$

where $\delta(x_V, x_{V_n})$ is the indicator function and $m(x_V) = \sum_n \delta(x_V, x_{V_n})$ denotes the number of times x_V was encountered in the dataset D. For the complete derivation of the log likelihood the interested reader is referred to [17], as it is omitted to preserve space. Taking exponentials to both sides of equation (6), we effectively express the probability distribution for D in the exponential family, with the counts $m(x_{\phi_u})$ being the sufficient statistics and $\log\theta_u(x_{\phi_u})$ the natural parameters. By maximizing (6) with respect to $\theta_u(x_{\phi_u})$, we derive the maximum likelihood estimate:

$$\hat{\theta}_{u,ML}(x_{\phi_u}) = \frac{m(x_{\phi_u})}{m(x_{\pi_u})} \tag{7}$$

In the presence of hidden variables in the graphical model (as it is the case with our approach – see section 3.3) or when we have *incomplete* observations (to account for the case of missing data values), we cannot use the maximum likelihood estimate directly to learn the conditional probability tables of the hidden variables. For that purpose we use the *EM* algorithm [7].

The algorithm performs a gradient ascent on the parameter space Θ, by alternating between the equations:

$$p^{(t+1)} = \arg\max_p L(p, \theta^{(t)}) \tag{8}$$

$$\theta^{(t+1)} = \arg\max_\theta L(p^{(t+1)}, \theta) \tag{9}$$

a maximum number of times or until convergence. Here, equation (8) corresponds to the *Expectation* step of the algorithm, equation (9) to the *Maximization* step, and $L(.)$ the utility function defined as:

$$L(p,\theta) =< l_c(\theta; x, z) >_p - \sum_z p(z|x)\log p(z|x), \tag{10}$$

with $< l_c(\theta; x, z) >_p$ denoting the expectation of the complete log likelihood with respect to the distribution p, and with z iterating over the hidden random variables.

We should note that, as it has been pointed out at section 3.3, we regard the conditional probability tables of the observed nodes given their corresponding hidden parent, $p(x_O|x)$, as known quantities, which remain unaltered between the EM iterations.

4.2 Structure Learning

Although it would seem that our overview of BN learning is presented in an unorthodox manner, as the order between the two tasks of structure and parameter learning is reversed in this section, it is often the case, as we shall see in the description of the SEM algorithm (paragraph 4.2), that a parameter estimation step is implied for the approximation of the optimal structure of the network, given the observed data. In the literature we can find two families of structure learning algorithms for BN's. The *Constraint-Based* family directs an iterative procedure of edge removals from a fully connected directed acyclic graph, whenever certain conditional independencies are measured in the data. A representative of this family is the PC algorithm [30]. According to the more popular *Search-and-Score* approach, a search in the space of all possible directed acyclic graphs is taking place, with the goal to find one that maximizes a criterion over the training data.

For the purposes of our work, we experimented with three algorithms of the latter family, which are presented in the following paragraphs. As an evaluation metric for the quality of the learned graphs, we used the Bayesian Information Criterion (BIC). Let G be the structure of a BN and D a training dataset. The BIC score of G with respect to D is given by:

$$BIC(G,D) = \log P(D|G, \hat{\theta}_{ML}) - \frac{1}{2} Dim(G)\log|D|, \tag{11}$$

where $Dim(G)$ is the network's dimension. For a definition of $Dim(G)$ see [21]. If we were to attempt a quick interpretation of the BIC score, we notice that it is based on the likelihood of the derived BN given the training data. The second term penalizes the score, encouraging the selection of "simpler" graphical models, in an attempt to limit overfitting phenomena. In that sense, it is equivalent to the Minimum Description Length (MDL) principle [22].

Exhaustive Search A relatively straightforward idea to implement the *Search-and-Score* methodology in the presence of complete observations is to perform an exhaustive search in the space of directed acyclic graphs (DAG's). This brute force approach, although it guarantees that it will find the best DAG to fit the training data, has a prohibitive (super-exponential) computational complexity for graphs with many nodes, and therefore it is not practically viable for $|V| > 5$. As Leray et al. suggest in [21], the number of DAG's to be evaluated as a function of the number of nodes is given by:

$$r(n) = \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{i} 2^{i(n-i)} r(n-i) = n^{2^{O(n)}} \quad (12)$$

K2 [3]. K2 implements a greedy search in the space of DAG's. Initially, all the nodes are disconnected. The search space is traversed by connecting each node u to the parent node that its addition to the family of u increases the score of the resulting structure. When the addition of no single parent can increase the score, the family for that node is fixed and the additions stop.

The algorithm is based on the assumptions that it is run on a complete dataset, the samples provided are *iid*, the prior probability of all structures is the same and that a specific node ordering has been provided as an input to speed up its convergence. Since PCET is agnostic of any heuristics or hints that might indicate conditional independence assumptions between the sensors which are modeled as random variables in the BN, we have no reason to favor any particular node ordering for K2's initialization. Therefore, we implemented a slight modification of the algorithm that executes it for user provided number of times, where at each execution we initialize it with a random node ordering. Finally, we pick the DAG with the highest score.

In the original publication of K2 [3], Cooper and Herskovits propose as an evaluation metric of a DAG's quality its probability over the training data:

$$P(D|G) = \prod_{i=1}^{n} \prod_{j=1}^{q_i} \frac{(r_i - 1)!}{(m(x_{\{ij\}}) + r_i - 1)!} \prod_{k=1}^{r_i} m(x_{\{ijk\}})! \quad (13)$$

where $n = |V|$, $r_i = |x_i|$, $q_i = |x_{\pi_i}|$ - that is q_i corresponds to the number of unique instantiations of the parent set of node i, and $m(x_{\{ijk\}})$ the number of times the corresponding instantiation of node i and its parent set occur in the data. Equation (13) is called the *Bayesian Measure* of the network. In [15], it has been proved that the Bayesian Measure is not *likelihood-equivalent*. Therefore, as a scoring metric, we use BIC score, as it is the score with which we evaluate both the exhaustive and the SEM algorithms.

Structural EM So far we have presented methods that elicit the BN's structure from complete training data sets. When we attempt to do so from *incomplete* observations, either because of missing data values for some observations or because of the existence of hidden random variables in the BN, the above approaches seize to apply. The main difficulties one has to face come from the fact that in order to evaluate a candidate BN, the necessary scoring metrics (usually variants of the log likelihood of the model, as the BIC criterion we use) do not decompose easily compared to the completely observed case, and therefore approximations of the parameters of the hidden variables have to be calculated, which imply the need of performing inference on the candidate BN (as the EM algorithm requires for parameter estimation). Secondly, a local change in one part of the network may affect the evaluation of a change in another part of it. Thus, a naive approach should need to perform an EM-style parameter estimation for each neighbouring network structure in the search space, leading to computationally expensive algorithmic solutions.

Friedman, in [12, 13] suggests an algorithm, namely STRUCTURAL EM (SEM), that reduces this problem to the complete observations case. Instead of approaching the problem in the naive manner mentioned above, SEM moves the traversal of the search space of the possible structures *inside* the E-step of the EM procedure, where it operates on the expected complete log likelihood setting. A pseudocode for the SEM algorithm follows:

```
Choose $M^0$ and $\Theta^0$ randomly
Iterate until convergence
    Find a model $M^{n+1}$ that maximizes $Q(\cdot|M^n, \Theta^n)$
    Let $\Theta^{n+1} = \arg\max_{\Theta} Q(M^{n+1}, \Theta|M^n, \Theta^n)$
```

where M is a model belonging to the class of models in which we are searching for the optimal with respect to the data and Θ is a parameter vector, such that each legal choice of values defines a probability distribution $P(\cdot|M, \Theta)$ over the set of random variables modeled in the BN. Finally, Q stands for a network's score metric. For a more theoretical presentation of the algorithm and proofs of convergence, see [12].

As expected, SEM's sensitivity to local maxima and initial conditions is more pronounced as opposed to the simple *em* version for parameter estimation. In [13] several heuristics are discussed (various random initializations, perturbations of the learned structure at each step, associations of hidden nodes to all the observed ones, etc.) to help the algorithm converge to a more global solution. In our implementation, we repeat the procedure a user-defined number of times, picking different random network structure initializations each time. We have also slightly modified the algorithm to learn BN's that follow the model presented in section 3.3. Therefore, we preserve the connections between the observed and their corresponding hidden variables, thus limiting the local search steps of the algorithm only between the various interconnections of the hidden nodes with each other.

4.3 Inference

The general problem that inference algorithms on BN's try to solve is the calculation of the conditional probability of a node or a set of nodes, given the observed values of another set of nodes. These algorithms have been extensively reviewed in the literature. Roughly, they can be categorized into those that focus on algebraic operations, such as VARIABLE ELIMINATION and the relatively more sophisticated BUCKET ELIMINATION [5], and those that focus on the graphical properties of BN's. A very efficient algorithm of the second category is the JUNCTION TREE algorithm [20], highly valued in the Statistical Learning community, since it enables the calculation of the requested conditional probabilities for all nodes simultaneously.

Despite its advantages in computational efficiency, the JUNCTION TREE algorithm cannot be considered as a generally applicable solution to the inference problem, as it is viable only for graphs with small treewidth[2]. Moreover, it is not clear from the JUNCTION TREE framework how to extend this technique in order to support the efficient conditional probability calculation of an arbitrary subset of nodes and not just of single or pairs of adjacent nodes in the graph.

Since in PCET the generated BN's structure cannot be controlled and since the user can pose queries (high-level events) that will eventually be mapped to arbitrary subsets of nodes, the advantages of the JUNCTION TREE can't be exploited. Instead, we chose the BUCKET ELIMINATION algorithm, which has higher computational complexity with respect to the number of nodes, but it is general enough to suffice for our needs. For more details on the specifics of this algorithm, see [5].

[2] The *treewidth* of a graph G is defined as the size of the maximum clique in the optimal triangulation of G, minus 1.

5 Implementation

As described in section 3.1, PCET is composed of two fundamental architectural elements: the Event Processing Engine (EPE), responsible for the generation of an query plan, based on the high level event definitions and its execution, and the Probabilistic Inferencing Engine (PIE), which calculates the probabilities of the events contained in a query occurring, based on the graphical model it maintains and the noisy sensor readings at that particular point in time. PIE has been implemented in MATLAB. It uses Kevin Murphy's BAYES NET TOOLBOX (BNT) [23], an open-source library that supports construction and inference over a large variety of static and dynamic Bayesian Networks. We experimented with the various structure, parameter learning and inference algorithms provided, and we extended the ones that we ended up using, namely Exhaustive Search, K2 and SEM, in order to learn the type of the BN described in section 3.3. EPE has been implemented in Java. It uses ANTLR (`http://www.antlr.org/`), a language recognition tool for generating the parser for PCET's SIMPLE EVENT LANGUAGE (see section 3.2).

Since PIE's operation is based on BNT, we had to resolve the problem of remote code execution in the MATLAB environment. Up to the latest version, MATLAB does not support a fully functional interface with Java applications which are executing in Java Runtime Environments other than the one MATLAB provides. In order to bridge this connectivity "chasm" with the rest of our system (namely with the EPE component), we resorted to the client-server communication model. We built a server program in Java, able to accept certain classes of MATLAB commands, execute them in MATLAB and return the result to the caller via sockets. This server java process "lives" inside the JRE that MATLAB hosts, and thus it can interact with its environment through a primitive and totally undocumented API. On the other side, a client class was written to pose commands and intercept results, which are forwarded to the EPE component.

Another difficulty we had to overcome was that the marshalling between MATLAB and Java complex data types (e.g. arrays, tables, etc.) is virtually non existent. That limited the flexibility of the PIE component to return only primitive data types. As a consequence, a large part of the inference engine's state had to be cached inside the MATLAB environment, while being controlled by the part of PIE residing in the client side. This design decision resulted in an inevitable increase of the communication overhead, since a lot of small transfers have to take place between these two runtime environments. We are still investigating alternative ways of using MATLAB's Java API, but since there is no documentation provided, we expect to progress at a relatively slow pace.

Overall, the system works as follows: Probabilistic queries (that is high level event definitions) are bulk loaded to generate an execution plan that translates them to operations between base-level events, which in turn correspond to instances of the schema associated with the sensors (the measured quantities of our world). These queries are registered to the PIE component, which is responsible of calculating their probabilities whenever new sensor readings arrive to the system. Since PCET is not fed by a continuous queries processing engine, we use a data abstraction layer, which PIE polls periodically to receive "fresh" data values. After the calculation of a query's probability, the result is pushed to the EPE component, to decide, based on the event's definition, if it has to be triggered to the application layer or not.

The training of the graphical model is currently performed off-line, with data generated by a simulator we developed for evaluation purposes. A description of the simulation component we used is provided in section 6.1. We expect that if we integrate at some point our system with a Continuous Query Processor (a quite natural extension to think about), we will be able to maintain the graphical model periodically, in a dynamic fashion, as data are fed into the event processing engine.

As far as the support we currently provide for the SIMPLE EVENT LANGUAGE is concerned, time constraints did not allow us to implement the aggregation operators and the time sequencing predicates, which we hope to support in a future version of our system.

6 Evaluation

In order to evaluate the robustness of our system with respect to the noise induced at the sensor readings, and to verify our assertions about the enhanced quality of the probabilistic inference we support by using the proposed graphical model, we performed a series of experiments, based on a particular application scenario in the context of the DIGITAL HOME project. In section 6.1 we describe the application scenario and the data generator we developed that simulates the sensor readings we would expect to accommodate it. Section 6.2 describes our experimental set-up and discusses the results of our evaluation.

6.1 Simulator

The data used to evaluate our system was produced using a simulator written for this purpose. We made this decision for a number of reasons:

- Real sensor data is generally difficult to obtain;
- A real data set contains only observed values, not actual values;
- A simulator allows us to examine scenarios that are as simple or as complicated as we want;
- In a simulator we can tweak various environmental parameters such as the amount of noise.

The simulator tries to reproduce with some degree of accuracy the interactions between people and the sensors found in a digital home. Source of our inspiration was the DIGITAL HOME project, currently under development at the Intel Research laboratory at Berkeley. Its main goal is to produce a useful, extensible software platform ("Home OS") for the efficient management of data streams coming from various sensing devices in the home environment; such a platform can, in turn, enable intelligent learning applications, e.g., for automated actuator control.

The input to the simulator is a scenario consisting of the layout of a house, a number of sensors, a number of people, and behavioral models for those people. The output is a set of sensor readings taken at regular intervals as the simulation is carried out and the virtual people move about. These sensor readings, which are 100% accurate with respect to the simulated environment, are then passed through a noise inducer that adds a variable amount of noise to the data. The noise inducer is modeled as a standard noisy channel in which the probability of interference is set according to a parameter defined by the scenario.

In order to understand how readings are generated, we must first introduce the basic framework upon which scenarios are constructed. Its components are described below.

Tiles Tiles are square regions of space that make up the simulated world. They are 2-dimensional only and have size but no position (or placement). Tiles are connected to each other forming undirected connected graphs. In order to model tiles that are mounted diagonally with respect to each other, each edge in the graph stores a factor (typically either 1 or $\sqrt{2}$). The distance between the centers of two adjacent tiles is proportional to the sum of their sizes and the edge factor. Tiles are usually used to model rooms by placing several in a grid where each inner tile connects to its 8 neighbors. Tiles are also used to model doorways and the space in front/behind the house. Certain tiles can also contain objects (TV, stove, etc.).

Sensors There are currently three types of sensors: motion, presence, and identification. A more complicated simulator would contain other types of sensors, such as audio, video, and thermal. Motion and presence sensors have a binary output (0 or 1), while ID sensors output the identifier of the person they detect (by RFID, visual, audio, or other means) or 0 if nobody is detected. Each sensor is hooked up to one or more tiles, which is a simple way of modeling the detection range of that sensor. A motion sensor is triggered when someone moves across one of the tiles it monitors at least once during a sampling interval. Similar rules apply to presence and ID sensors, with the exception that an ID sensor has to randomly pick a person if more than one are detected in a sampling period.

People People are modeled as point-like entities in our 2D world. Each person has its own average speed but it can move a bit slower or faster (at random). There are currently only two actions a person can perform: walk to a given location or stand still. When walking to a destination tile, people always use the shortest path according to the tile graph. Every time they walk across a tile, they will trigger any sensors attached to it. In contrast, when standing still only presence and ID sensors are triggered.

Tasks Tasks encapsulate the behavioral model of a person. They are roughly equivalent to procedures in a program and can be arranged in a tree, with the main task at the root. Tasks correspond to the motivations of individuals, and they prompt people to perform specific actions. Whenever a person finishes an action, the root task is consulted to get the next action to perform (during this process a subtask may be consulted). The only task implemented in our simulator is the random action task. As the name implies, it causes the individual to either stand still or walk to a random location of interest. Locations of interest are defined by the scenario (the center of each room, for instance) and the random choice is done using a weighted lottery algorithm. In a more complicated simulator, people would have state (such as hunger, boredom, energy) and there would be subtasks aimed at "optimizing" their state (e.g. by making them go to the kitchen to eat when hungry).

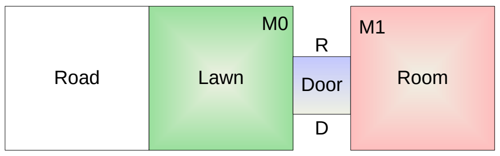

Fig. 4. The sensor deployment of the "Simple Scenario".

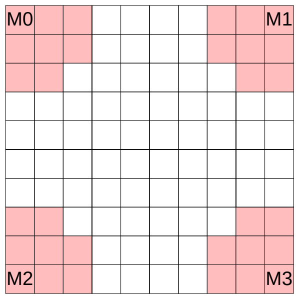

Fig. 5. The sensor deployment of the "Four Corners Scenario".

We have implemented two scenarios to generate data for our system. The first that we call "Simple Scenario" contains a one-room house, a person, and four sensors of different kinds rigged to detect comings and goings, namely two motion sensors, one inside and the other outside the room, a pressure sensor at the doorstep inside the room and an RFID tag reader inside the room also (see figure 4). The second one, named "Four Corners", contains a room with no doors, 4 motion sensors mounted in the corners, and a lone occupant (see figure 5). In both scenarios it is possible to adjust the amount of expected correlation among the sensor readings via a tunable parameter. In "Simple Scenario", the parameter controls the anxiety of the simulated people; turning it up makes them pause less often and causes them to pace back and forth between outside and inside. In "Four Corners", the parameter controls the range of the motion detectors, which can be increased from one tile until each sensor covers the entire room (complete correlation).

6.2 Experimental Results

Model Evaluation In order to evaluate the different structures we obtained from the various structure learning algorithms we experimented with (Exhaustive Search, K2 and SEM), we run a series of experiments, with generated training data from the simulator of the "Simple Scenario" described in the previous section (4 discrete-valued sensors which are modeled by an 8-node BN– see section 3.3 for the specifics of the model used), varying the noise level and the training set size. To evaluate the inferred networks from the training data, we used the BIC score, described in section 4.2.

Due to time constraints, we were unable to produce an indicative set of results, as the amount of training time for K2 and SEM, especially with reasonable training set sizes of more than 100 samples, was prohibitive. As we mentioned in section 4.2, both K2 and SEM require random initializations in order to learn a good structure. K2 depends on the elimination ordering specified. To break this dependence, for each experiment we were learning 10 different BN's, each time by invoking K2 with a random elimination ordering and keeping the one with the highest BIC score.

This problem was exaggerated with SEM, since it is particularly sensitive to initial conditions also. As its creator suggests [12], the combined space of parameters and structure on which we perform the EM algorithm is filled with local maxima, thus finding the global maximum requires a lot of invocations of SEM with different random initial BN structures. Furthermore, in [12, 13] it is also suggested that in order to hinder SEM from returning as an optimal structure a network with the hidden nodes disconnected from all the rest, one has to initialize the BN by connecting each hidden node with all the observed ones, so as to increase the coupling between these nodes and the rest of the network, effectively forming a bipartite graph. While we followed the first suggestion (multiple invocations of the algorithm with random initializations), the special form of our model (each hidden node connected only to its corresponding observed node) did not permit us experimenting with the effects of the second one. Thus, we realized that when the sample size used was fairly small (under 30 instances), SEM would favor BN's with the hidden nodes disconnected from its other. The situation was

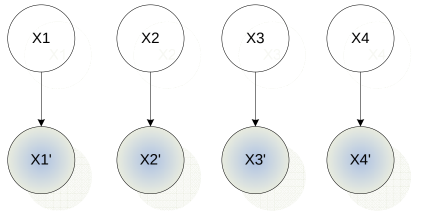

Fig. 6. The "All Independent" model, which follows the simplifying assumption that the sensor readings are independent of each other.

ameliorated when we increased the sample size, with the trade-off of a multiplicative increase in training time.

Overall, for one particularly meaningful experiment with 500 samples and for two different noise levels, we found both K2 and Exhaustive Search to produce very similar results, a phenomenon which was attributed to the fact that for such a simple network (only 4 sensors), both algorithms converged to the same structure, with SEM slightly outperforming the former two. This result was to be expected, since the technique we used to employ K2 and Exhaustive Search in an incomplete data training setting (see section 4.2 for further details) was only an approximation, compared to SEM, which despite its long training time and variability, can effectively learn a better structure from incomplete observations.

System Evaluation PCET was evaluated in terms of the "Simple Scenario" (section 6.1). We defined some high-level events comprised by logical operations on the readings of the four sensors of the simulated scenario, which we fed into the system. Each event was accompanied by a probabilistic threshold, according to which a notification should be triggered if the probability of that event happening, given the current sensor readings, exceeded that threshold.

Due to time constraints, we used only K2 to infer the structure of the BN model, which we compared to the "All Independent" model, representing the simplistic approach followed to date in the relative literature (see figure 6). SEM was excluded from these experiments due to its large training time, and so was Exhaustive Search, not only because it produced similar graph structures with K2, but also because of its exponential computational complexity, which makes it impractical for larger networks. To evaluate the robustness of our model towards noise, we varied the noise level induced to the sensors, from 0.1 up to

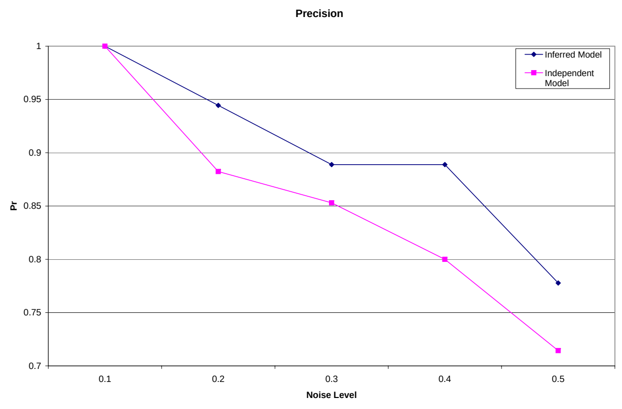

Fig. 7. The Precision of the Independent vs. the Inferred from the data BN, for various noise levels.

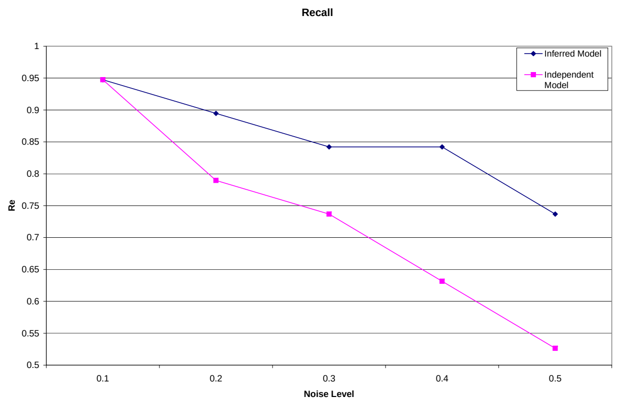

Fig. 8. The Recall of the Independent vs. the Inferred from the data BN, for various noise levels.

an extend that it would still be meaningful to rely on the sensors' readings (0.5)[3]. The training set size for all the experiments was fixed at 100 samples.

The evaluation metrics used were Precision, Recall and Accuracy, well respected inside the Information Retrieval community. Precision $\left(Pr = \frac{TP}{TP+FP}\right)$ serves as a degree of "soundness" of the system, corresponding to the conditional probability of the occurrence of an event, given that it has been triggered by PCET. Recall $\left(Re = \frac{TP}{TP+FN}\right)$, a degree of "completeness", corresponds to the conditional of an event being triggered by PCET given that it actually occurred, reflecting the fraction of the actual events PCET managed to identify. Finally, Accuracy $\left(Acc = \frac{TP+TN}{TP+TN+FP+FN}\right)$ tries to combine the above measures, giving a unified view of the performance of the system.

In figures 7 through 9 we present the experimental results, for the five different noise levels used. As noise levels increase, performance deteriorates for both models, with the model inferred by K2 always outperforming the independent one. While Precision seems to degrade at an equivalent rate for both, Recall of the independent model is severely affected by the presence of noise. This is not the case for the K2 model, since the correlations of the various sensors encoded by the graphical model's structure help PCET recognize more actually occurring events, despite the fair amount of noise that distorts the sensors' readings. Thus, through the BN we minimize the number of False Negative results we obtain, which are directly associated with the Recall measure, a phenomenon which is intuitively reasonable. On the other hand, by increasing the noise levels, we make both models vulnerable to False Positive identifications of non occurring events, which explains why the Precision measure deteriorates more rapidly.

[3] In the noise scale we used, 1.0 corresponds to the point where half of the times the sensor returns the actual value of the sensed quality, while the rest of the readings are distorted with noise.

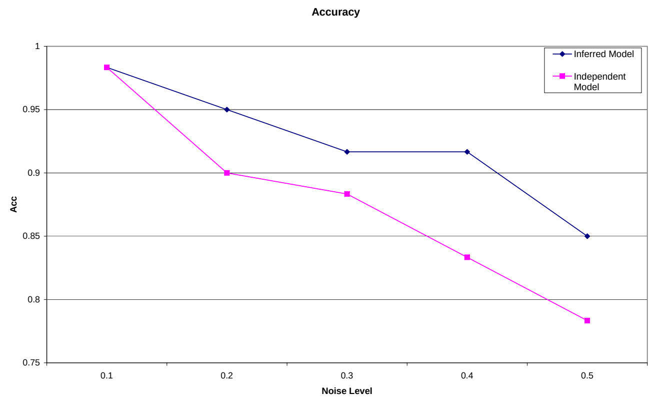

Fig. 9. The Accuracy of the Independent vs. the Inferred from the data BN, for various noise levels.

These encouraging results confirm our assertions about the beneficial effect of modeling the correlations of readings observed in a sensornet deployment, stimulating various interesting research directions. As it was mentioned often within this section, time was our biggest enemy. Nevertheless, we plan to continue evaluating our infrastructure, by testing it on larger scale scenarios, by evaluating the performance of SEM and by investigating the effect of varying the correlations between the different sensors. Some first steps towards the later goal have already been achieved (see section 6.1), but considerable amount of work needs to still be done in order to support this extension.

7 Conclusions - Future Work

In this paper we presented PCET, an infrastructure that attempts to resolve the problem of robust event detection and triggering in the presence of noisy inputs, while at the same time offering an expressive language through which probabilistic event hierarchies can be easily specified. The end result of this effort is a prototype of an event processing system, able of performing probabilistic inference on complex events provided as input.

We evaluated PCET in the context of the DIGITAL HOME project, currently under development at the Intel Research laboratory at Berkeley, arguing that it can serve as a "central nervous system", a core processing unit, through which the event flow of all the sensing devices in the home will be processed and the necessary stimuli will be dispatched to the corresponding applications. The results from the simple people tracking application scenario we experimented with confirm that a probabilistic framework can greatly enhance the robustness of an event processing system towards noisy data input, and that the latter goal can be achieved without sacrificing the expressiveness which the hierarchical structure of event definition and processing gives to the application developer.

This paper represents our initial work in the field of probabilistic event processing, and as such it introduces a number of interesting, worth exploring future directions. These can be categorized in the terms of further enhancing the functionality, the efficiency and the expressiveness of our system:

Functionality – Due to time constraints, we limited the probabilistic modeling and inferencing of the correlations among the sensors to exploring the discrete data case only. An interesting direction would be to extend our modeling framework in order to support continuous data as well. This raises the challenges of incorporating both discrete and continuous data in the same BN and of choosing (or inferring) the appropriate continuous distribution that fits the data.
- A shortcoming of our implementation is the simulation of a data layer that feeds the system with the raw event streams, as they are generated from the sensors. An interesting extension would be the integration of PCET's engine with an existing continuous query processor [2].

Efficiency – One part of our system that deems for further work is the statistical learning component. The BUCKET ELIMINATION inference algorithm we use, although general enough, exhibits prohibitive computational overhead to be of practical use. Furthermore, of particular interest is the problem of efficient BN structure inference from incomplete data. The SEM algorithm, although being computationally expensive, represents a pioneering approach in this intriguing line of research, upon which fresher ideas are currently built [9]. A closer investigation of those approaches could be beneficial for both the quality of the inferred models and their generation time.
- Clearly, the redirection of all the queries referring to the probability of a complex event down to the probabilistic model for the base events is expected to become a serious bottleneck of our system. A possible scheme we could use to address this problem is the construction of a probabilistic model for the events of each layer in the hierarchy. This approach is inspired from the view materialization literature, and thus it shares the same trade-offs (efficiency vs view updating).
- Another consideration involves multi-query optimization issues. Since at each level a number of queries are generated for the calculation of the joint probability of a subsets of random variables, such calculations might be shared among probabilistic queries.

Expressiveness – Our current implementation supports only a subset of the expressiveness of the language described in section 3.2, namely basic logical operators and their counterparts that support probabilistic calculus over them. The implementation of the aggregation operator could enhance the expressiveness of the high level event definitions, as it would give our system the total of the expressive power and flexibility that other event processors known to date [32, 6] share. Furthermore, a very useful extension would be the implementation of the time sequencing operator, which will enable the end-users to pose queries that associate a temporal ordering on the events to be monitored.

– Taking this idea a step further, once could argue that the incorporation of time into the definition of an event seems a rather complicated issue, since it will involve the learning of associations not only among different events but between the occurrence of specific event patterns in time. For instance, although it seems perfectly natural to correlate temperature readings with respect to the time of day they occur and upon those to form events with temporal constraints, the probabilistic models that will have to be maintained, if implemented naively, will grow exponentially in number. The relative literature has to show a number of interesting approaches on Dynamic Bayesian Networks, some of which are located in the relational context (see [28]), that deserves a more thorough investigation, as to how they can help extend the graphical model we are currently using.

References

1. D. Barbara, H. Garcia-Molina, and D. Porter. The management of probabilistic data. *IEEE Transactions on Knowledge and Data Engineering*, 4(5):487–502, 1992.
2. Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel R. Madden, Fred Reiss, and Mehul A. Shah. Telegraphcq: Continuous dataflow processing. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 668–668, New York, NY, USA, 2003. ACM Press.
3. Gregory F. Cooper and Edward Herskovits. A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9:309–347, 1992.
4. Nilesh N. Dalvi and Dan Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.
5. Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In Eric Horvitz and Finn Jensen, editors, *Proceedings of the 12th Conference on Uncertainty in Artificial Intelligence (UAI-96)*, pages 211–219, San Francisco, August 1–4 1996. Morgan Kaufmann Publishers.
6. Alan Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riederwald, and Walker White. A general algebra and implementation for monitoring event streams. In *Submitted for publication*, 2005.
7. A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.
8. A. Deshpande, C. Guestrin, S. Madden, J. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the 30th VLDB Conference*, Toronto,Canada, 2004.
9. Gal Elidan, Noam Lotner, Nir Friedman, and Daphne Koller. Discovering hidden variables: A structure-based approach. In *NIPS*, pages 479–485, 2000.
10. Anton Faradjian, Johannes Gehrke, and Philippe Bonnet. GADT: A probability space ADT for representing and querying the physical world. In *ICDE*, 2002.
11. Michael J. Franklin, Shawn R. Jeffery, Sailesh Krishnamurthy, Frederick Reiss, Shariq Rizvi, Eugene Wu, Owen Cooper, Anil Edakkunni, and Wei Hong. Design considerations for high fan-in systems: The hifi approach. In *CIDR*, pages 290–304, 2005.
12. Nir Friedman. Learning belief networks in the presence of missing values and hidden variables. In *Proc. 14th International Conference on Machine Learning*, pages 125–133. Morgan Kaufmann, 1997.
13. Nir Friedman. The bayesian structural em algorithm. In *Proc. 14th International Conference on Artificial Intelligence*, pages 129–138. Morgan Kaufmann, 1998.
14. Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.
15. David Heckerman, Dan Geiger, and David Maxwell Chickering. Learning bayesian networks: The combination of knowledge and statistical data. In *KDD Workshop*, pages 85–96, 1994.
16. Shawn R. Jeffery, Gustavo Alonso, Michael J. Franklin, Wei Hong, and Jennifer Widom. A pipelined framework for online cleaning of sensor data streams. In *ICDE*, 2006.
17. Michael Jordan. *An Introduction to Probabilistic Graphical Models*. 2006.
18. Kidd, Cory, Orr, Robert, Gregory D, Abowd, Gregory, Atkeson, Christopher, Essa, Irfan, MacIntyre, Blair, Mynatt, Elizabeth, Starner, Thad, Newstetter, and Wendy. The aware home: A living laboratory for ubiquitous computing research. In *Proceedings of the Second International Workshop on Cooperative Buildings (CoBuild 1999)*, 1999.
19. Laks V. S. Lakshmanan, Nicola Leone, Robert Ross, and V. S. Subrahmanian. ProbView: a flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, 1997.
20. S. Lauritzen and D. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, 50:157–224, 1988.
21. Philippe Leray and Olivier Francois. Bnt structure learning package: Documentation and experiments. Technical Report FRE CNRS 2645, Laboratoire PSI, 2004.

22. Tom M. Mitchell. *Machine Learning.* 1997.
23. Kevin Murphy. The bayes net toolbox for matlab. *Computing Science and Statistics*, 33, 2001.
24. Matthai Philipose, Kenneth P. Fishkin, Mike Perkowitz, Donald J. Patterson, Dieter Fox, Henry Kautz, and Dirk Hahnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3(4):50–57, 2004.
25. Luc De Raedt and Kristian Kersting. Probabilistic logic learning. *SIGKDD Explor. Newsl.*, 5(1):31–48, 2003.
26. Sira Panduranga Rao and Diane J. Cook. Predicting inhabitant action using action and task models with application to smart homes. *International Journal on Artificial Intelligence Tools*, 13(1):81–99, 2004.
27. Shariq Rizvi, Shawn R. Jeffery, Sailesh Krishnamurthy, Michael J. Franklin, Nathan Burkhart, Anil Edakkunni, and Linus Liang. Events on the edge. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 885–887, New York, NY, USA, 2005. ACM Press.
28. Sumit Sanghai, Pedro Domingos, and Dan Weld. Relational dynamic bayesian networks. *Journal of Artificial Intelligence Research*, 24:759–797, 2005.
29. Anish Das Sarma, Omar Benjelloun, Jennifer Widom, and Alon Halevy. Working models for uncertain data. In *ICDE*, 2006.
30. P. Spirtes, C. Glymore, and R. Schines. *Causation Prediction, and Search.* 1993.
31. Jennifer Widom and Stefano Ceri. *Active Database Systems: Triggers and Rules For Advanced Database Processing.* Morgan Kaufmann, 1996.
32. D. Zimmer and R. Unland. On the Semantics of Complex Events in Active Database Management Systems. In *Proceedings of the 15th International Conference on Data Engineering*, pages 392–399. IEEE Computer Society Press, 1999.